INDEX

- GPU-friendly networks
  - history: APEnet+
  - research: GGAS, PEACH2,
  - corporate: w/ MLNX OFED 1.5.4.1+GDR, OFED 2.1/2.2
- GDR enabler for low-perf CPUs
- current limitations and WARs
  - accelerate eager for GPU
- perspective: I/O bus limited -> NVLINK
SCALING PROBLEM

- many interesting apps bang on the scaling wall
  - single big problem = capability computing
  - fixed physical problem size
  - parallel decomposition
  - lots of GPUs $O(10^2-10^3)$
- different reasons
  - unbalance
  - poor coding
  - overheads -> GPU<->network interaction
- even optimized apps
  - let’s move the wall further away...
GPU-friendly networks
RESEARCH

P2P/GPUDirectRDMA on APEnet+

- APEnet+ card:
  - Network Processor, SoC design
  - FPGA based
  - 8-ports switch
  - 6 bidirectional links (max 34 Gbps)
  - PCIe X8 Gen2 in X16 (4+4 GB/s)
  - 32bit RISC

- Accelerators:
  - Zero-copy RDMA host interface
  - GPU P2P support

[*] R.Ammendola et al, “GPU peer-to-peer techniques applied to a cluster interconnect”, CASS 2013
DIRECT VS (PIPELINED) STAGING

- Staging to host memory:
  - 14us latency
  - 7-10us penalty for each cudaMemcpy
  - < 1us NIC HW latency
  - SW complexity
    - Pipelined algorithm
    - MVAPICH2, OpenMPI

Direct path:
- 7us latency (11/2011)
- GPU<->GPU path
  - both NV P2P and GPUDirectRDMA
- GPU<->3rd party devices
  - HPC scaling
  - Other applications: HEP, Astronomy
APENET

2013: APENet+ Bandwidth (PCIe Gen2 X8, Link 30Gbps, Westmere Bridge)

Bandwidth (MB/s)

Message size (32B-4MB)

APENet+ GPU-to-GPU Roundtrip Latency (Westmere Bridge + C2050)

Latency (us)

Message size (32B-4KB)
GPU-NETWORK FUSION CARD

- Frankenstein board
  - Take Half of a “GTX 590”
  - Add dedicated APEnet
- Tune net bandwidth
- Put 1,2,.. per server
- Perfect for HPC
RESEARCH

PEACH2 & GGAS

- PEACH2 @tsukuba
- GGAS=Global GPU Address Spaces @Heidelberg

- Communication model based on global address spaces & shared memory accesses
- Well-know from the past
  - Suffers from scalability issues due to consistency/coherence
- Reverting to highly relaxed consistency models can be a solution
  - Note that GPUs use a very weak consistency model
GPUDIRECT RDMA ON MELLANOX IB BOARDS

- GPUDirect RDMA support
  - early prototype started in Feb 2013
  - presented at GTC 2013 on MVAPICH2 1.9
- rework for Mellanox SW stack
  - May 2013 - Dec 2013
  - showcased at SC’13 at Mellanox booth
  - shipping since 1/2014 MOFED 2.1
  - OpenMPI 1.7.4
  - MVAPICH2 2.0b
GPUDIRECT RDMA + INFINIBAND + CUDA-AWARE MPI

Less overhead = lower latency and more effective bandwidth

When GDRDMA is effective:

- 3x more BW
- ~70% less latency

Not always good:

- Experimentally, cross-over behavior at 8-16KB
- GDRDMA reading BW cap at 800MB/s on SNB
- GDRDMA writing BW saturates.

Preliminary Performance of MVAPICH2 with GPU-Direct-RDMA
Cross-over Between GPU-Direct RDMA and Host Pipelining

2 nodes MPI benchmarks with and without GPUDirect RDMA
Intel SNB, K20c, MLNX FDR PCIe X8 gen3
ON IVY BRIDGE XEON

MVAPICH2 D-D Latency (K40m PCIe X16 Gen3, ConnectX-3 PCIe X8 Gen3, IVB E5-2690v2 @3GHz)

- D-D GDR only
- D-D staging

MVAPICH2 D-D Bandwidth (K40m PCIe X16 Gen3, ConnectX-3 PCIe X8 Gen3, IVB E5-2690v2 @3GHz)
MPI USING GPUDIRECT RDMA

back in GTC14 time...

- any space for improvement?
BENCHMARK RESULTS

- MV2-GDR 2.0b original version
- MV2-GDR-New experimental version

Graph showing latency for different message sizes. The graph compares latency between MV2-GDR 2.0b, MV2-GDR-New (Loopback), and MV2-GDR-New (Fast Copy). The graph highlights a 52% reduction in latency when comparing MV2-GDR-New (Fast Copy) to MV2-GDR 2.0b. Specific latency values are not shown in the text. The graph includes data for CUDA 6.0, Mellanox OFED 2.1 with GPUDirect-RDMA plug-in.
RESTORING EAGER

recalling Eager protocol

Eager protocol on IB:

- **sender:**
  - copy on pre-registered TX buffers
  - `ibv_post_send`, on UD, UC, RC, ...

- **receiver:**
  - pre-post temp RX buffers, use credits for proper bookkeeping
  - RX matching then **copy** to final destination
RESTORING EAGER

Eager essential tool for low-latency / small msgs

get rid of rendezvous as:

- bad interplay with apps
- excessive sync among nodes
- more round-trip’s

Problem:

- staging: moving data from host temp buf to GPU final dest
  - cudaMemcpy has >4us overhead (~24KB threshold at 6GB/s)
RESTORING EAGER

Two tricks

- IB loopback
  - use NIC as low-latency copy engine

- GDRCopy
  - CPU-driven zero-latency BAR1 copy
RX IB LOOPBACK FLOW

TX side

RX side
EAGER WITH GDRCOPY

(ab)use CPU to copy data to/from GPU mem

- experimental super low-latency copy library
- three sets of primitives:
  - Pin/Unpin, setup/tear down BAR1 mappings of GPU memory
  - Map/Unmap, memory-map BAR1 on user-space CPU address range
    - CPU can use standard load/store instructions (MMIO) to access the GPU memory.
  - Copy to/from PCIe BAR, highly tuned R/W functions
MPI RX GDRCOPY FLOW

TX side

src_buf
GPU MEM
GPU
MLNX C-IB
CPU MEM
CPU
GPU MEM
dst_buf
CPU
MLNX C-IB
GPU

GDRCopy data path

RX side

post buf
CPU
CPU MEM
GDRCOPY

performance

- zero latency vs ~1us for loopback vs 4-6us for cudaMemcpy
- D-H: 20-30MB/s
- H-D: 6GB/s vs 9GB/s for cudaMemcpy
- sensitive to
  - CPU MMIO performance
  - NUMA effects, eg on IVB D-H=3GB/s when on wrong socket

available soon on https://github.com/drossetti/gdrcopy
# BENCHMARK RESULTS

on IVB Xeon + K40m + MLNX C-IB

<table>
<thead>
<tr>
<th>#</th>
<th>MV-2.0b</th>
<th>MV2-GDR-2.0b</th>
<th>MV2-GDR</th>
<th>loopback</th>
<th>gdrcopy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.27</td>
<td>1.31</td>
<td>1.16</td>
<td>1.22</td>
<td>1.21</td>
</tr>
<tr>
<td>1</td>
<td>19.23</td>
<td>7.03</td>
<td>5.29</td>
<td>4.77</td>
<td>3.19</td>
</tr>
<tr>
<td>2</td>
<td>19.26</td>
<td>7.02</td>
<td>5.28</td>
<td>4.78</td>
<td>3.18</td>
</tr>
<tr>
<td>4</td>
<td>19.35</td>
<td>7.01</td>
<td>5.26</td>
<td>4.77</td>
<td>3.17</td>
</tr>
<tr>
<td>8</td>
<td>19.45</td>
<td>7.00</td>
<td>5.26</td>
<td>4.79</td>
<td>3.17</td>
</tr>
</tbody>
</table>

* with MVAPICH2 GDR pre-release
**BENCHMARK RESULTS**

**bandwidth**

**Performance of MVAPICH2 with GPUDirect-RDMA: BW & Bi-Bandwidth**

Intel Ivy Bridge... Message Bi-Bandwidth

Message Size (Bytes)

Bi-Bandwidth (MB/s)

2.1x

2.2x
coming soon...
MULTI-GPU APPS SCALING PROBLEM

Time Spent in Subroutines

- Compute kernels scale
- Communication cost is constant

MVAPICH2 2.0b
K20X GPU
64000 particles
HOOMD-BLE BLUE PROFILING

MV2-GDR 2.0b

CPU idle

GPU BAR1 here but invisible

MPI Launch latency

Process 'hoomd_liquid'
Thread 26205873535
Runtime API
Driver API
Markers and Ranges

Profiling Overhead
10 Tesla K40m
Context 1 (CUDA)
Memcpy (HtoD)
Memcpy (DtoH)
Compute
Streams
Default
Stream 23
LIMITED SCALING

how it works today

- CPU mgmt, eg cudaStreamQuery() & MPI_Test():
  - poll GPU kernels and trigger NIC comm
  - poll NIC comm then schedule GPU kernels

- Today scaling needs fast CPU
  - implications for ARM
  - slow CPU not enough at ~ 10^2 GPUs
1D STENCIL COMPUTATION

halo<<<stream0>>>(h0, p0);
MPIX_Isend(p0, stream0, &req0[0]);
halo<<<stream1>>>(h1, p1);
MPIX_Isend(p1, stream1, &req[0]);
bulk<<<stream2>>>(d);
MPIX_Irecv(h0, stream0, &req0[1]);
MPIX_Wait(stream0, req0, 2);
cudaEventRecord(e0, stream0);
MPIX_Irecv(h1, stream1, &req1[1]);
MPIX_Wait(stream1, req1, 2);
cudaEventRecord(e1, stream1);
cudaStreamWaitEvent(stream2, e0);
cudaStreamWaitEvent(stream2, e1);
Energy<<<stream2>>>(d);
cudaStreamSynchronize(stream2);

Wait for both Isend and Irecv

Even record will wake the stream wait event

expand into CUDA + IBverbs + verbs/CUDA interop
DEPENDENCY GRAPH

stream0
- Halo krn
  - Isend
    - Bulk krn
      - Irecv
        - Wait
          - event record
            - Stream event wait
              - Stream event wait
                - Energy krn

stream2
- Halo krn
  - Isend
    - Irecv
      - Wait
        - event record
          - Stream event wait

stream1
- Halo krn
  - Isend
    - Irecv
      - Wait
        - event record
          - Stream event wait

Command Engine
Compute Engine

Kepler Workflow
- Stream Queues
  - Ordered queues of grids
  - CLUDA-Generated Work
    - Pending & suspended grids
      - 1000's of pending threads
        - Two-way link allows pausing a batch
          - Work Distributor
            - Actively dispatching grids
              - SMX
                - SMX
                  - SMX
                    - SMX
NEXT GEN I/O

NVlink 1.0 in Pascal

- beyond PCIe
- additional data path, 80-200GB/s
- NVlink attached network?
SUMMARY

- p2p, platforms (IVB) still limited, Haswell?
- challenge, manage multiple datapaths and topologies, eventually NVLINK
- reduce overheads, use GPU scheduler
Thank You !!!

Questions ?
PCI BAR?

- **BAR**=Base Address Register
- PCI resource
- up to 6 32bits **BAR** regs
- 64bits **BAR** uses 2 regs
- physical address range
- **prefetchable** ~ cachable [danger here]
- most GPUs expose 3 **BARs**
- **BAR1** space varies:
  - **K20**/40c=256MB
  - **K40m**=16GB [new!]

```bash
$ lspci -vv -s 1:0.0
```

```
01:00.0 3D controller: NVIDIA Corporation Device 1024 (rev a1)
  Subsystem: NVIDIA Corporation Device 0983
  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
  Stepping= SERR= FastB2B= DisINTx-
  Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
  <MAbort- >SERR- <PERR- INTx-
  Latency: 0
  Interrupt: pin A routed to IRQ 86
  Region 0: Memory at fa000000 (32-bit, non-prefetchable) [size=16M]
  Region 1: Memory at b0000000 (64-bit, prefetchable) [size=256M]
  Region 3: Memory at c0000000 (64-bit, prefetchable) [size=32M]
```

```bash
$ nvidia-smi
```

```
BAR1 Memory Usage
...```
GPU BAR

BAR1 = aperture on GPU memory

GPUDirectRDMA == GPU BAR1 as used by 3rd party devices:

- network cards
- storage devices
- SSD
- reconfigurable devices
- FPGAs
PLATFORM-RELATED LIMITATIONS

PCIE topology and NUMA effects:

- number and type of traversed chipsets/bridges/switches
  - GPU memory reading BW most affected
    - Sandy Bridge Xeon severely limited
  - On older chipsets, writing BW affected too
- crossing inter-CPU bus huge bottleneck
  - currently allowed though
NUMA EFFECTS AND PCIE TOPOLOGY

- GPU BAR1 reading is half PCIE peak, by design

- GPUDirect RDMA performance may suffer from bottlenecks
  - GPU memory reading most affected
  - old chipsets, writing too
  - IVB Xeon PCIE much better
use GPUDirectRDMA for small/mid size buffers

- threshold depends on platform
- and on NUMA (e.g. crossing QPI)

don’t tax BAR1 too much

- bookkeeping needed [being implemented]

revert to pipelined staging through SYMEM

- GPU CE latency >= 6us

no Eager-send for G-to-G/H
PLATFORM & BENCHMARKS

- SMC 2U SYS-2027GR-TRFH
  - IVB Xeon
  - PEX 8747 PCIE switch, on a raiser-card
- SMC 1U SYS-1027GR-TRF
- NVIDIA K40m
- Mellanox dual-port Connect-IB

Benchmarks:
- GPU-extended ibv_ud_pingpong
HOST TO GPU (IVY BRIDGE)

TX side

GK110B

x16 Gen3

MLNX C-IB

x16 Gen3

SYSMEM

IVB Xeon

BW: 6.1GB/s
lat: 1.7us

RX side

GK110B

Single rail FDR 6.1 GB/s

MLNX C-IB

Single rail FDR 6.1 GB/s

SYSMEM

IVB Xeon
GPU TO HOST (IVY BRIDGE)

RX side

GK110B

MLNX C-IB

SYSMEM

BW: 3.4/3.7*GB/s
lat: 1.7**us

TX side

GK110B

MLNX C-IB

SYSMEM
GPU TO GPU (IVY BRIDGE)

TX side

GK110B

MLNX C-IB

Single rail FDR 6.1 GB/s

x16 Gen3

SYSMEM

IVB Xeon

RX side

GK110B

MLNX C-IB

Single rail FDR 6.1 GB/s

x16 Gen3

SYSMEM

IVB Xeon

BW: 3.4/3.7*GB/s
lat: 1.9us
GPU TO HOST (SANDY BRIDGE)

RX side

GK110 (x16 Gen2) → MLNX C-X3 (x8 Gen3)

SYSMEM → SNB Xeon

TX side

GK110 (x16 Gen2) → MLNX C-X3 (x8 Gen3)

SNB Xeon → SYSMEM

Single rail FDR 6.1 GB/s

BW: 800MB/s
GPU TO GPU, TX ACROSS QPI

TX side

GK110B

x16 Gen3

MLNX C-IB

Single rail FDR 6.1 GB/s

IVB Xeon

BW: 1.1GB/s
lat: 1.9us

RX side

GK110B

x16 Gen3

MLNX C-IB

Single rail FDR 6.1 GB/s

IVB Xeon

SYSMEM
GPU TO GPU, RX ACROSS QPI

RX side

GK110B (x16 Gen3) → MLNX C-IB (x16 Gen3) → IVB Xeon → MLNX C-IB → GK110B

Single rail FDR 6.1 GB/s

TX side

GK110B (x16 Gen3) → MLNX C-IB (x16 Gen3) → IVB Xeon → MLNX C-IB → IVB Xeon → SYSMEM

Single rail FDR 6.1 GB/s

BW: 0.25 GB/s
lat: 2.2us
HOST TO GPU, RX ACROSS PLX

TX side

GK110B

X16 Gen3

MLNX C-IB

X16 Gen3

PLX

SYSMEM

IVB Xeon

RX side

GK110B

MLNX C-IB

PLX

IVB Xeon

Single rail FDR 6.1 GB/s

Single rail FDR 6.1 GB/s

BW: 6.1 GB/s

lat: 1.9us
GPU TO GPU, TX/RX ACROSS PLX

TX side

GW110B

MLNX C-IB

PLX

X16 Gen3

X16 Gen3

IVB Xeon

RX side

GW110B

MLNX C-IB

PLX

X16 Gen3

X16 Gen3

IVB Xeon

Quick facts:

- Single rail FDR 6.1 GB/s
- BW: 5.8GB/s
- lat: 1.9us